最小进程(Minimal process)

NtCreateProcessEx函数设置PROCESS\_CREATE\_FLAGS\_MINIMAL\_PROCESS标志

(也就是这两行代码

PROCESS\_CREATE\_FLAGS flags = { 0 };

flags.MinimalProcess = 1;）

并且其调用方位于内核模式时，该函数的行为会略有差异，并会导致执行PsCreateMinimalProcess这个API，进而导致进程在创建时缺乏很多结构，主要为：

1. 不在设置用户模式地址空间，因此不存在PEB和相关结构。
2. 进程不会有映射的ntdll.dll，也不会有任何加载程序/API集的信息。
3. 进程不会绑定区域对象(section object)，也就意味着进程的执行或其名称（为空或任意字串）不具备相关的可执行映像文件。
4. EPROCESS标志中会设置Minimal标志，导致所有线程成为最小线程，同时可以避免任何用户模式的分配，例如TEB结构或用户模式栈。

Windows10至少自带2个最小进程：System和Memory Compression进程，如果启动基于虚拟化的安全性，可能还会有第三个最小进程Secure System进程。

在 Windows 10 中，启用基于虚拟化的安全性可以通过以下步骤完成：

1. 打开“控制面板”并选择“程序”。
2. 在“程序和功能”下，选择“打开或关闭 Windows 功能”。
3. 在“Windows 功能”窗口中，向下滚动并找到“虚拟化基础平台”选项。勾选该选项。
4. 点击“确定”保存更改并等待系统完成设置。

完成上述步骤后，可以使用 Windows Defender Application Guard、Credential Guard 和 Device Guard 等功能来增强系统的安全性。(并非所有硬件和BIOS都支持)

还可以在Windows 10中启动适用于Linux的Windows子系统(WSL)这一可选功能。运行最小程序，借此即可安装有Lxss.sys和LxCore.sys驱动程序组成且系统自带的Pico提供程序。

Pico进程

在允许从内核组件访问用户模式虚拟地址空间并对其进行保护方面，最小进程(Minimal process)的用途较为有限。Pico进程则可以让一种名为Pico提供程序(Pico Provider)的特殊组件控制(从操作系统的角度看)其执行过程中的大部分操作。这种程度的控制最终使得此类提供程序可以模拟与操作系统内核截然不同的行为，并确保底层用户模式库无法察觉到自己在Windows中运行。从本质上来说，这是微软研究院Drawbridge项目的一种实现，也被用于通过类似方式实现让SQL Server支持Linux(尽管Linux内核的基础上使用了一种基于Windows库的操作系统。

为了让系统支持Pico进程，先得有提供程序，提供程序可以注册至PsRegisterPicoProvider API，但是Pico提供程序必须先于先于任何第三方驱动程序加载(包括引导驱动程序)。实际上，在有限的十几个核心驱动程序中，只有一个可以在该功能被禁用前调用这个API，且这些核心驱动程序必须使用微软签名方证书和Windows组件EKU签名。在启用了可选WSL组件的Windows系统中，此驱动为Lxss.sys。可在LxCore.sys稍后加载之前充当存根(stub)驱动程序，随后将由LxCore.sys接管Pico提供程序，进而将多种分派表转移给自己。

LxCore.sys和Lxss.sys（来自chatGPT）

LxCore.sys是Windows Subsystem for Linux (WSL) 的内核模块，它充当WSL系统的核心，提供了WSL所需的基础设施，如Linux系统调用转换、文件系统映射和内存管理等。而Lxss.sys是Windows Subsystem for Linux 的会话模块，它是WSL与用户模式Linux子系统之间的接口，提供了WSL系统和Linux子系统之间的通信，以及对Linux系统调用的转换和处理。

如何利用这两个驱动加载Pico进程（来自chatGPT,不一定靠谱）

在 Windows 10 中，LxCore.sys 和 Lxss.sys 驱动程序是用于启动 WSL 2 Pico 进程的。要加载 Pico 进程，需要按照以下步骤进行操作：

1. 安装 WSL 2 并确保 LxCore 和 Lxss 驱动程序已安装。可以通过检查 Windows 设备管理器中的系统设备来确认这一点。
2. 使用 **CreateProcess** 或 **CreateProcessW** 函数启动一个新进程。在 **STARTUPINFOEX** 结构体中，将 **lpAttributeList** 成员设置为指向 **PROC\_THREAD\_ATTRIBUTE\_PSEUDOCONSOLE** 属性的指针。**PROC\_THREAD\_ATTRIBUTE\_PSEUDOCONSOLE** 属性将终端控制台句柄与进程相关联，使其能够与控制台进行交互。
3. 在进程创建后，使用 **InitializePseudoConsole** 函数初始化伪终端。此函数会将伪终端输入输出流关联到 Pico 进程的标准输入输出流。
4. 使用 **SetConsoleMode** 函数设置控制台输入输出模式，以便将控制台的输入和输出重定向到伪终端。

当Pico提供程序调用注册API时，会收到一系列函数指针，进而借此创建并管理Pico进程，这些指针分别为：

1. 创建Pico进程的函数，创建Pico线程的函数
2. 获取Pico进程的上下文的函数，设置Pico进程上下文的函数(可为EPROCESS 的PicoContext字段填充数据）
3. 获取Pico线程的上下文的函数，设置Pico线程上下文的函数（可为ETHREAD的PicoContext字段填充数据）
4. 获取Pico线程的CPU上下文（CONTEXT结构）的函数，设置Pico线程CPU上下文的函数
5. 用于更改Pico线程FS或GS段的函数，这些段通常被用户模式代码用于指向某些线程本地结构(如TEB结构)
6. 用于终止Pico线程的函数，用于终止Pico进程的函数
7. 用于挂起Pico线程的函数，用于挂起后恢复Pico线程的函数

Pico提供程序可以借助这些函数创建完整的自定义进程和线程，并自行控制初始启动阶段、段注册以及相关数据。但这用来模拟一个操作系统还不够，需要转移另一组函数指针，将提供程序转移至内核，当Pico线程或进程执行某些值得关注的活动后充当回调函数，这些函数说明如下：

1. 当Pico线程使用SYSCALL指令(用户模式进内核的方法之一)发起系统调用时对应的回调函数。
2. 当Pico线程抛出异常时的回调函数。
3. 当Pico线程内部针对内存描述符列表(Memory Descriptor List, MDL)执行深测和锁操作出错时的回调函数

关于内存描述列表和深测和锁操作：(chatGPT)

对内存描述符列表执行深度搜索和锁操作（Deep Search and Lock on Memory Descriptor List）是一种在Windows内核开发中常用的技术，用于在内存描述符列表中查找特定的内存区域，并在找到目标内存后对其进行加锁，以避免其他线程或进程对其进行修改或释放。

内存描述符列表（Memory Descriptor List）是一种数据结构，用于跟踪和管理系统中的物理内存。在Windows内核开发中，开发人员通常需要在内存描述符列表中搜索或遍历特定的内存区域，以执行各种操作，例如内存映射、内存拷贝等。由于多个线程或进程可以同时访问内存描述符列表中的内存区域，因此需要使用锁来保护这些区域，以避免竞争条件和数据损坏。(其对应**EPROCESS** 结构体中的 **MmProcessLinks** 字段。该字段是一个 **LIST\_ENTRY** 结构体类型的变量，表示了一个双向链表，用于管理进程所拥有的内存描述符对象。每个内存描述符对象是一个 **MM\_SESSION\_SPACE** 结构体类型的变量)

深度搜索和锁操作是一种比较安全和可靠的内存访问方式，它可以保证对目标内存区域的安全访问和操作，但是由于需要对整个内存描述符列表进行搜索和锁定，因此效率较低，不适合对大量内存区域进行操作。

1. 当调用方请求Pico进程名称时的回调函数
2. 当Windows事件跟踪(ETW)请求Pico进程用户模式栈跟踪时的回调函数
3. 当应用程序试图打开Pico进程或Pico线程句柄的回调函数
4. 当请求终止Pico进程时的回调函数
5. 当Pico线程或Pico进程意外终止时的回调函数。

Pico提供程序还可以利用内核补丁保护(Kernel Patch Protection, KPP)机制保护自己的回调和系统调用。

显然，借助这种前所未有的访问用户-内核间可能的转换，以及Pico进程/线程和其余系统组件间可见的内核-用户交互，Pico提供程序（以及相关的用户模式的库）可以通过完整封装的方式对Windows之外其他完全不同的内核实现进行彻底的包装（当然还有一些例外，线程调度规则和“提交”等内存管理规则依然适用）

Pico提供程序本质上是一种自定义开发的内核模块，通过实现必要的回调响应Pico进程可能产生或抛出各种可能事件。因此WSL能够在用户模式下运行未经修改的Linux ELF二进制文件，这一过程仅受制于系统调用模拟工作和相关功能的完善程度